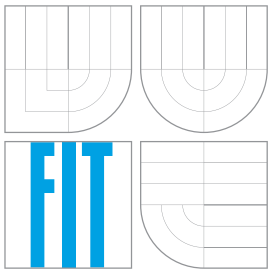
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

DATOVÁ STRUKTURA BLOOMŮV FILTR A JEJÍ POUŽITÍ PRO SMĚROVÁNÍ V INTERNETU

BLOOM FILTERS AND THEIR USE IN INTERNET ROUTING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PAVEL VRAŠTIAK
AUTHOR

VEDOUCÍ PRÁCE — Ing. VIKTOR PUŠ
SUPERVISOR

BRNO 2010

Abstrakt

Tato práce se zabývá datovou strukturou Bloomův filtr a jeho praktickou aplikací při směrování dat v Internetu. Budou zde popsány základní vlastnosti této datové struktury a pokusíme se ukázat, proč by vyžití Bloomova filtru mohlo přinést zajímavé výsledky při operaci hledání nejdelšího shodného prefixu. Implementační část práce tvoří softwarovou implementaci tohoto algoritmu v jazyce C.

Abstract

This thesis is considering Bloom filter data structure to be used in Internet routing. We will describe properties of this data structure and explain why Bloom filters could bring great results in longest prefix matching operation. Algorithm is implemented in C language.

Klíčová slova

Bloomův filtr, směrování, TCP/IP, vyhledání IP adresy, rozptylovací tabulka

Keywords

Bloom filter, routing, TCP/IP, IP lookup, hash table

Citace

Pavel Vraštiak: Datová struktura Bloomův filtr a její
použití pro směrování v Internetu, bakalářská práce, Brno, FIT VUT v Brně, 2010

Datová struktura Bloomův filtr a její použití pro směrování v Internetu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Viktora Puše. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Pavel Vraštiak
10. května 2010

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu práce za podnětné rady a ochotu, kterou projevoval během tvorby této práce.

Obsah

1 Úvod **3**

2 IP sítě **4**
2.1 Architektura TCP/IP 4
2.2 Principy směrování v IP sítích 6

3 Datová struktura Bloomův filtr a její základní vlastnosti **8**
3.1 Chyby prvního a druhého druhu 8
3.2 Popis algoritmu 8
3.3 Počítaný Bloomův filtr 9
3.4 Časová a prostorová složitost 9
3.5 Pravděpodobnost výskytu chyby prvního druhu 10
3.6 Příklady praktického použití 11

4 Použití Bloomových filtrů při vyhledání nejdelšího shodného prefixu **12**
4.1 Důsledek chyb prvního druhu 13
4.2 Proč je důležité omezit počet přístupů do paměti? 14

5 Konfigurace a optimalizace **15**
5.1 Asymetrické (nestejné) Bloomovy filtry 15

6 Popis implementace **18**
6.1 Parametry programu 18
6.2 Formát vstupu a výstupu 19
6.3 Knihovna pro práci s rozptylovací tabulkou 20
6.4 Bloomův filtr 20
6.5 Vyhledání nejdelšího shodného prefixu 21

7 Výsledky **22**
7.1 Paměťová složitost 22
7.2 Počet přístupů do rozptylovacích tabulek 23
7.3 Srovnání počtu zbytečných přístupů 26

8 Další možná vylepšení **27**
8.1 Redukce filtrů reprezentující krátké prefixy 27
8.2 Další snížení počtu filtrů 29

9 IPv6 **30**
9.1 Architektura IPv6 adres 31
9.2 Přidělování adres 31

10 Závěr **32**

A Obsah DVD **34**

Kapitola 1

Úvod

Práce se zabývá Bloomovými filtry a jejich použití při směrování, konkrétně při technice vyhledání nejdelšího shodného prefixu.

V kapitole 2 jsou nejprve popsány základní principy směrování v IP sítích a vysvětleny základní pojmy, jako je *směrovací tabulka* či *next hop* adresa. Kapitola 3 nabízí popis datové struktury Bloomův filtr a vysvětluje principy jeho základních operací. Část 4 popisuje, jak je možné využít Bloomovy filtry pro operaci vyhledání nejdelšího shodného prefixu.

Dále je v kapitole 5 popsána asymetrická konfigurace filtrů, která by měla přinést lepší vlastnosti celého algoritmu. V kapitole 6 je popsána implementace, rozhraní programu a formát vstupních i výstupních dat. Následující kapitola 7 se zabývá vyhodnocením výsledků experimentů. Navazující kapitola 8 naznačí některá možná vylepšení, které povedou k omezení nejhoršího možného případu nadbytečných přístupů do paměti.

Předposlední kapitola 9 ukazuje, že téměř všechny zde uvedené principy platí i pro IPv6 a naznačí odlišnosti, které je třeba mít na paměti při práci s touto vyšší verzí protokolu IP.

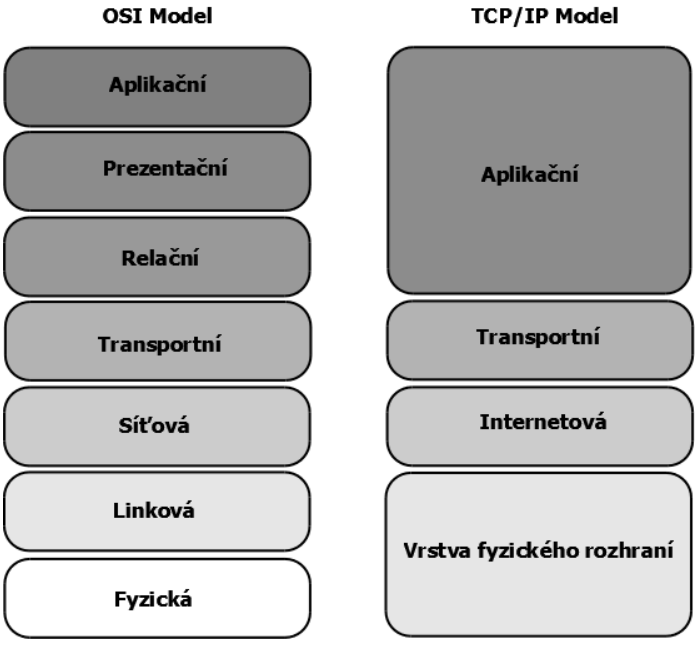
Kapitola 2

IP sítě

Pro popis síťové architektury se používá referenční model OSI (Open Systems Interconnect Reference Model) definovaný Mezinárodní standardizační organizaci ISO (International Standards Organization) v roce 1987 jako rámec standardů pro komunikaci po síti mezi různými zařízeními a aplikacemi různých výrobců [9]. Z referenčního modelu je odvozen model TCP/IP, který je základem Internetu a reflektuje strukturu této sítě.

2.1 Architektura TCP/IP

Architektura TCP/IP je výrazně jednodušší než u referenčního modelu OSI. Model TCP/IP spojuje služby prezentační a relační do jediné vrstvy, aplikační. Na úrovni fyzického přenosu bitů spojuje vrstvy fyzickou a linkovou do jedné vrstvy, vrstvy fyzického rozhraní, která je obvykle implementována na síťové kartě. Porovnání obou modelů je ukázáno dále na Obrázku 2.1.



Obrázek 2.1: Souvislost mezi referenčním modelem OSI a TCP/IP modelem.

Tento návrh obecně odstraňuje nedostatky modelu OSI, jehož komplikovaná struktura vrstev nebyla nikdy plně implementována, především díky přílišné složitosti.

Popis jednotlivých vrstev tohoto modelu:

1. **Vrstva fyzického rozhraní (Network Access Layer)** popisuje standardy a protokoly pro dané fyzické medium. Definuje elektrické signály a jejich interpretaci přijímající stranou, dále zahrnuje funkce pro přístup k fyzickému médiu a zajišťuje také zabalování IP datagramů do rámců. K adresování na této vrstvě slouží fyzická adresa. U technologií založených na Ethernetu (802.3, 803.11) je tato adresa 48-bitová MAC adresa. Fyzická adresa se používá pro adresování rámců v lokálních sítích.

2. **Internetová vrstva (IP Layer)** vytváří IP datagramy, zajišťuje jejich adresování a směrování na místo určení. Snaží se najít nejlepší cestu pro dosažení cílového zařízení. Internetová vrstva kromě protokolu IP (Internet Protocol) pro přenos a směrovaní datagramů používá také protokoly ARP (Address Resolution Protocol) a RARP (Reverse ARP) pro mapováni IP adres na MAC adresy, dále protokol ICMP (Internet Control Message Protocol) pro řízení toku a detekci nedosažitelných uzlů, či protokol IGMP (Internet Group Management Protocol) pro přihlašovaní se do multicastových skupin. Tyto protokoly musí byt součástí implementace IP modulu.

   Základní operace IP vrstvy jsou:

   √ definice datagramů a způsobu adresování
   √ přenos dat mezi internetovou vrstvou a fyzickým rozhraním
   √ směrovaní datagramů na vzdálený počítač

   K identifikaci síťového zařízení na internetové vrstvě TCP/IP modelu resp. síťové vrstvě v OSI modelu slouží IP adresa. Tato adresa je ve verzi 4 protokolu IP 32-bitová, u verze 6 dokonce 128-bitová.

3. **Transportní vrstva (Transport Layer)** provádí přenos dat od zdrojového počítače na cílový. Vytváří logické spojeni mezi komunikujícími procesy. Transportní protokoly rozdělují aplikační data na menší jednotky - pakety, které jsou posílány po síti. Základními protokoly jsou TCP a UDP:

   - TCP (Transmission Control Protocol) načítá data z aplikační vrstvy jako proud dat, který seskupuje do číslovaných paketů. Pakety posílá v určeném pořadí k cíli. TCP zajišťuje spolehlivý přenos dat - řízení toku, pořadí paketů, potvrzování a řízení zahlcení (angl. congestion control).
   - UDP (User Datagram Protocol) je označován jako tzv. nespojovaná služba. Nezajišťuje navázání spojení, číslování a potvrzování paketů, tudíž je rychlejší než TCP. Případná komunikace musí počítat s možnými ztrátami paketů, přičemž ani jejich pořadí doručení není zaručeno.

   Na transportní vrstvě jsou adresovány služby, které běží na koncových zařízeních, jako e-mailový server, služba DNS apod. Adresu služby přenáší ve své hlavičce TCP či UDP protokol ve formě 16-bitového čísla portu. Tato hodnota identifikuje službu (aplikační proces), která data posílá (číslo portu odesilatele), a přijímající aplikační proces (číslo portu adresáta).

4. **Aplikační vrstva (Application Layer)** je tvořena procesy a aplikacemi, které komunikují po síti. Zajišťuje zpracováni dat na nejvyšší úrovni včetně reprezentace dat, jejich kódovaní i řízení dialogu. Adresování na aplikační úrovni závisí na konkrétní aplikaci.

2.2 Principy směrování v IP sítích

Směrování představuje způsob hledání cesty v síti pro odchozí pakety a využívá jej každé zařízení připojené do IP sítě. I koncové počítače, byť v jejich případě bývá směrování triviální. Jeho úkolem je dopravit datový paket určenému adresátovi, pokud možno co nejefektivnější cestou. Síťová infrastruktura mezi odesílatelem a adresátem paketu může být velmi složitá. Směrování se proto zpravidla nezabývá celou cestou datagramu, ale řeší vždy jen jeden krok - komu data předat jako dalšímu. Každý odeslaný IP datagram je opatřený informací o cílové síti. Když IP diagram dorazí ke směrovači, tak ten se podívá do své *směrovací tabulky* (routing table) a odešle tento datagram na příslušné rozhranní. Směrovací tabulka představuje vlastně sadu ukazatelů, podle kterých se rozhoduje, co udělat s příchozími datagramy. Příklad směrovací tabulky:

Network	Next Hop	Metric	Path
169.232.0.0/16	137.164.130.61	1	1164 2152 52
224.16.0.0/16	137.164.130.57	20	1164 2152 52
192.168.18.0/24	137.164.130.53	20	1164 2152 52
10.0.0.0/16	213.248.98.93	48	1299 356 252 2152 52
10.10.0.0/16	64.214.121.169	49	3549 209 252 2152 52

Tabulka 2.1: Příklad jednoduché směrovací tabulky

Směrovací tabulka je především složena ze záznamů obsahujících následující:

- **Cílová IP adresa.** Může se jednat o adresu individuálního počítače, častěji však je cíl definován prefixem, tedy začátkem adresy. Prefix mívá podobu 147.229.0.0/16. Hodnota před lomítkem je adresa sítě, hodnota za lomítkem pak určuje počet významných bitů adresy. Uvedenému prefixu tedy vyhovuje každá adresa, která má v tomto konkrétním případě v počátečních 16ti bitech hodnotu 147.229.
- IP Adresa následujícího skoku (**Next hop**) nebo jméno rozhraní, kam bude datagram poslán. Ten může být doručen přímo adresátovi nebo předán některému ze sousedních zařízení.
- "Cena"cesty, **metrika**. V případě více shodných cest bude vybrána ta s lepší metrikou.

Směrovací rozhodnutí pak probíhá samostatně pro každý přicházející datagram. Vezme se jeho cílová adresa a porovná se se směrovací tabulkou následujím způsobem:

1. Z tabulky se vyberou všechny vyhovující záznamy (jejichž prefix vyhovuje cílové adrese datagramu).
2. Z vybraných záznamů se použije ten s nejdelším shodným prefixem.
3. Podle způsobu směrování může být při nenalezení shodujícího se záznamu v tabulce použita výchozí brána nebo může být datagram zahozen.

Cesty v tabulce jsou buď statické nebo dynamické. Statické záznamy jsou vytvořeny ručně správcem sítě, kdežto dynamické jsou tvořeny automaticky, ale vyžadují použití a konfiguraci směrovacích protokolů.

Ty lze dále rozdělit na:

Distance vector protokoly
Směrovače udržují směrovací tabulku s informací o (vektoru) vzdálenosti do dané sítě, periodicky ji zasílají sousedům, ti si upraví svoji tabulku a tu opět odešlou dál, pro výpočet nejlepší cesty se používá jedna nebo více metrik - příkladem je RIPv1, RIPv2 nebo IGRP.

Link-state protokoly
Směrovače udržují komplexní databázi síťové topologie, sousední směrovače si prostřednictvím *"hello"* paketů vyměňují informace o stavu linek - zástupcem je protokol OSPF.

Path vector protokoly
Slouží ke směrování mezi autonomními systémy, které má charakteristické požadavky nevyskytující se v interním směrování. Směrovací tabulky obsahují stovky tisíc záznamů, nejdůležitějším kritériem nebývá vzdálenost, ale posuzují se nastavitelné parametry zohledňující například cenu a dodatečná pravidla aplikovaná v závislosti na zdroji, cíli, seznamu tranzitních autonomních systémů a dalších atributech - příkladem je protokol BGP.

Díky stále vzrůstajícímu počtu zařízení komunikujících v síti a nutnosti přidělení adresy každému tomuto zařízení, byla představena technika beztřídního adresování CIDR (*Classless Inter-Domain Routing*), jejímž úkolem bylo prodloužení životaschopnosti IPv4 protokolu. Pomocí beztřídního adresování není délka síťové části IP adresy určena třídou, s pevně danou délkou síťové části, nýbrž může mít libovolný počet bitů. Tento krok znamenal efektivnější využití adresového prostoru, zároveň nám na druhou stranu tento přístup komplikuje směrování, protože směrovače musejí prohlížet adresy s proměnnou délkou prefixu k tomu, aby zjistily nejdelší shodný prefix a následně získaly cílovou adresu pro každý procházející paket.

Tento krok však situaci zcela nevyřešil, a proto byl představen protokol IPv6, jehož adresy mají dé1ku 128 bitů, což teoreticky představuje 2^{128} rozdílných adres. Některé z těchto adres jsou, podobně jako u IPv4, rezervovány pro speciální účely, z čehož vyplývá, že reálně lze těchto adres využít o něco méně.

Kapitola 3

Datová struktura Bloomův filtr a její základní vlastnosti

Bloomův filtr byl navrhnut Burton H. Bloomem v roce 1970 [3]. Jedná se o paměťově nenáročnou pravděpodobnostní datovou strukturu, která slouží k dotazům na členství prvků v množině, kterou daný filtr reprezentuje. I přes to, že byly představeny možné náhrady Bloomova filtru [11], jeho jednoduchost, snadnost použití a dobré výsledky z něj dělají standardní datovou strukturu, která nalézá uplatnění v řadě aplikací. Díky pravděpodobnostnímu chování struktura připouští výskyt chyb prvního druhu (v angl. literatuře označovány *false positive*), avšak výskyt chyb druhého druhu (*false negative*) nikoliv.

3.1 Chyby prvního a druhého druhu

Chyba prvního druhu nastává, je-li hypotéza zamítnuta, přestože platí. A chyba druhého druhu nastává, pokud hypotéza zamítnuta není, přestože neplatí.

V kontextu této práce chyba prvního druhu nastane, pokud pro prvek, který do množiny reprezentované filtrem nepatří, nastanou takové podmínky, že jej chybně určíme jako prvek do této množiny patřící. A za chybu druhého druhu označíme situaci, pokud prvek, který do naší množiny patří, je odmítnut jako cizí prvek.

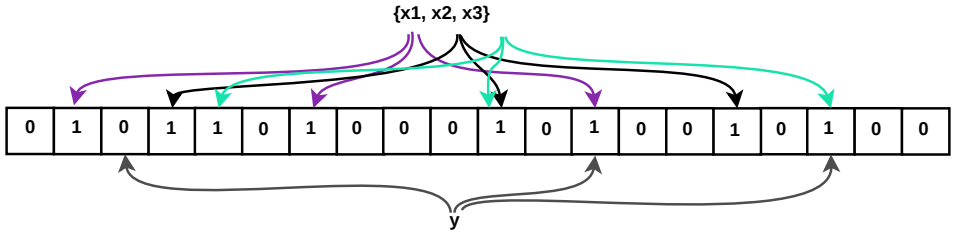
3.2 Popis algoritmu

Prázdný Bloomův filtr je tvořen bitovým polem o délce m bitů, všechny nastaveny na hodnotu 0. Dále filtr obsahuje k různých rozptylovacích funkcí produkujících k hodnot s náhodným rovnoměrným rozložením v rozsahu m, daným délkou bitového pole.

Mějme množinu $S = \{x_1, x_2, ..., x_n\}$ obsahující n prvků. Bloomův filtr reprezentující tuto množinu je vytvořen následujícím způsobem. Pro každý prvek x_i z množiny S jsou vypočteny hodnoty rozptylovacích funkcí. Každá z těchto hodnot adresuje pozici bitu v bitovém poli a každý z těchto bitů je nastaven na hodnotu 1. Je-li příslušný bit již nastaven, pak se jeho hodnota nezmění. Test na členství v množině spočívá ve výpočtu všech k rozptylovacích funkcí a poté v pouhém zjištění hodnot k bitů na pozicích danými hodnotami těmito funkcemi vypočtenými. Je-li alespoň jeden z nich nulový, pak prvek do množiny určitě nepatří, protože jinak by daný bit byl nastaven jednou z rozptylovacích funkcí. Naopak jsou-li všechny tyto bity nastaveny, pak prvek s jistou pravděpodobností p do množiny

patří. Pokud jsou tyto bity nastaveny a prvek do množiny S nepatří, jedná se o chybu prvního druhu.

Příklad Bloomova filtru představujícího tříprvkovou množinu s parametry $k = 3, m = 20$ je na obr. 3.1. Dotazovaný prvek y do množiny nepatří, protože jeden z bitů, na který se tento prvek mapuje, není nastaven:



Obrázek 3.1: Ukázka Bloomova filtru s parametry $k = 3$ a $m = 20$

Odebírání prvku z jednoduchého Bloomova filtru není umožněno. Každý prvek je namapován na k bitů, a přestože by k odebrání prvku zcela postačovalo nastavit libovolný z nich na hodnotu nula, mohla by tato operace odebrat i další prvky namapované na tento bit. Došlo by tak k zavedení chyby druhého druhu, což by porušilo základní principy této struktury, a proto je do jednoduchého filtru umožněno pouze prvky do množiny přidávat. Odebírání prvků však řeší počítané Bloomovy filtry.

3.3 Počítaný Bloomův filtr

Počítaný Bloomův filtr udržuje vektor čítačů korespondujících s každým bitem v bitovém poli filtru. Kdykoliv je prvek přidán resp. odebrán z filtru, jsou všechny hodnoty čítačů na pozicích danými výsledky k rozptylovacích funkcí zvýšeny, resp. sníženy. Nenulová hodnota čítače znamená, že odpovídající bit je nastaven. Je-li hodnota čítače změněna na nulu, je odpovídající bit také vynulován.

3.4 Časová a prostorová složitost

Na rozdíl od tradičních datových struktur, jakými jsou například pole, seznamy, samovyvažovací stromy či rozptylovací tabulky, má Bloomův filtr velice malou prostorovou složitost. Většina těchto struktur totiž vyžaduje ukládání alespoň dat samotných. Tyto data obecně mohou nabývat prakticky libovolných typů a velikostí, od jednoduchých celých čísel až po řetězce znaků. Ukazateli spojené datové struktury mají další požadavky na místo právě pro ukazatele.

Bloomův filtr s 1% dovolenou chybou a optimálním počtem rozptylovacích funkcí zabere pouze 9.6 bitu pro každý prvek - bez ohledu na jejich velikost:

$$f = 0.01 = \left(\frac{1}{2}\right)^{\frac{m}{n}\ln 2} \Rightarrow \frac{m}{n} = 9.59 \tag{3.1}$$

Tato vlastnost pramení částečně z jeho kompaktnosti zděděné z polí a částečně plynoucí z jeho pravděpodobnostní povahy. Každým přidáním průměrně 4.8 bitu na každý vložený

prvek lze chybu zmenšit zhruba desetkrát:

$$\frac{f}{\frac{f}{10}} = \frac{\left(\frac{1}{2}\right)^{\frac{M_1}{N}\ln 2}}{\left(\frac{1}{2}\right)^{\frac{M_2}{N}\ln 2}} \tag{3.2}$$

$$4.8N = (M_2 - M_1) \tag{3.3}$$

Je-li však počet potenciálních hodnot malý, pak může i obyčejné bitové pole vykazovat lepší výsledky než Bloomův filtr, protože požaduje pouze jeden bit na každý možný prvek. Další zajímavou vlastností Bloomova filtru je to, že čas potřebný k přidání prvku či k ověření jeho příslušnosti do množiny je konstantní - $O(k)$ a je zcela nezávislý na počtu prvků již do filtru vložených. Žádná jiná struktura s konstantní prostorovou složitostí nemá tuto vlastnost. Je pravdou, že některé řídké rozptylovací tabulky mohou vykazovat vyšší rychlost, avšak v hardwarové implementaci nabízí Bloomův filtr nespornou výhodu, protože jeho k vyhledávacích dotazů je nezávislých a mohou být provedeny paralelně.

Musíme však přijmout jistou pravděpodobnost výskytu chyby prvního druhu.

3.5 Pravděpodobnost výskytu chyby prvního druhu

Neboli pravděpodobnost, že pro dotazovaný prvek, který nebyl do filtru "vložen", budou hodnoty všech k bitů daných rozptylovacími funkcemi nastaveny. Pravděpodobnost, že náhodný bit bude rozptylovací funkcí nastaven je $\frac{1}{m}$. Pravděpodobnost, že bit nebude nastaven žádnou z k funkcí je zřejmě $\left(1 - \frac{1}{m}\right)^k$. Vložíme-li n prvků, pak pravděpodobnost, že daný bit je stále 0 je $\left(1 - \frac{1}{m}\right)^{kn}$. Z čehož vyplývá pravděpodobnost, že bit je nastaven do jedničky je doplňkem, tedy $1 - \left(1 - \frac{1}{m}\right)^{kn}$. Aby byl daný prvek označen jako možný člen našeho filtru, pak musí být všech k pozic bitového pole, daných rozptylovacími funkcemi, rovno hodnotě 1. Tato pravděpodobnost, označme ji f, je tedy:

$$f = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \tag{3.4}$$

Pro velké hodnoty m lze výše uvedený vztah aproximovat jako:

$$f \approx \left(1 - e^{\frac{-nk}{m}}\right)^k \tag{3.5}$$

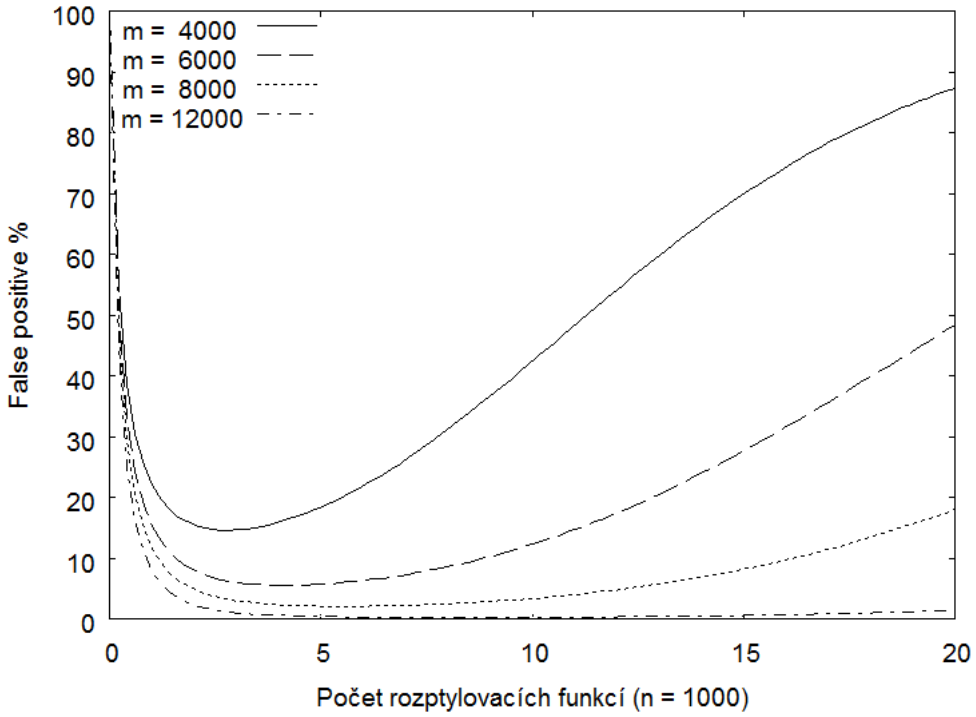
Z čehož vyplývá, že tato pravděpodobnost je závislá na velikosti filtru, počtu prvků v něm uložených a počtu použitých rozptylovacích funkcí a je možno ji snížit volbou vhodných parametr m a k pro danou n-prvkovou množinu S. V optimálním případě platí následující vztah pro počet rozptylovacích funkcí:

$$k = \frac{m}{n}\ln 2 \tag{3.6}$$

A pro tento optimální případ je hodnota této pravděpodobnosti dána vztahem:

$$f = \frac{1}{2^k} \tag{3.7}$$

Dále platí, že abychom pravděpodobnost výskytu chyby prvního druhu zachovali neměnnou, pak je potřeba zajistit, aby se velikost filtru, m, měnila přímo úměrně s velikostí množiny, n, pro kterou filtr tvoříme.



Obrázek 3.2: Závislost chyby na počtu rozptylovacích pro $n = 1000$ a různé parametry m

3.6 Příklady praktického použití

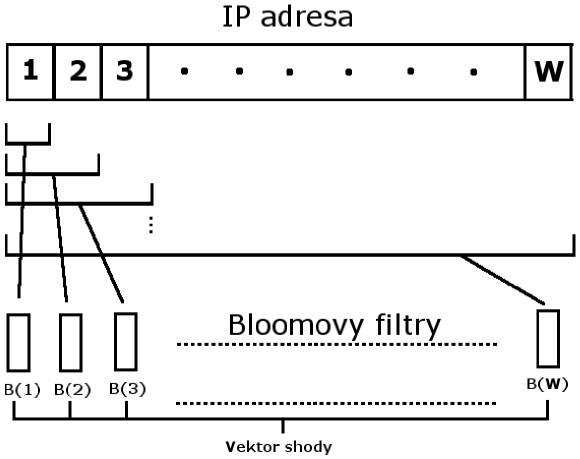
Mezi příklady nejstarší aplikace Bloomových filtru uvedených v [4] patří kontrola pravopisu v UNIXu, kde namísto prohledávání uloženého slovníku bylo použito Bloomova filtru pro reprezentaci slov ve slovníku obsažených. Bylo to především z důvodu nižší paměťové náročnosti tohoto způsobu. Dále lze zmínit třeba firmu Google, která implementuje Bloomův filtr ve své Big Table [5].

Kapitola 4

Použití Bloomových filtrů při vyhledání nejdelšího shodného prefixu

Technikám vyhledání nejdelšího shodného prefixu se v posledních letech dostalo značné pozornosti. Je to především z toho důvodu, že jsou základem směrování na směrovačích a přímo ovlivňují jejich výkonnost. Základem této operace je seskupení záznamů ve směrovací tabulce podle délky prefixu a přiřazení samostatného počítaného Bloomova filtru pro každou tuto délku zvlášť. Každý z těchto W filtrů je "naprogramován" pro danou množinu prefixů stejné délky. Pro každou odlišnou délku prefixu je navíc zkonstruována tabulka s rozptýlenými položkami (*rozptylovací tabulka*). Každá tato tabulka je inicializována příslušnou množinou prefixů, přičemž položky tabulky jsou tvořeny páry prefix, next hop adresa.

Vyhledávání spočívá v dotazu na členství vhodných částí vstupní IP adresy. Jednobitový prefix adresy je vstupem filtru přiřazeného délce jedna, dvoubitový prefix adresy je vstupem filtru, jež zpracovává prefixy délky dva atd. Každý z těchto filtrů oznámí, zda našel shodu, či nikoliv. Výsledkem sdružení výstupů od všech filtrů je vektor indikující možné shody pro příslušné délky prefixu pro danou vstupní IP adresu, přičemž platí, že některé z nich mohou být chybně určeny. Tomuto vektoru budeme říkat *vektor shody*.



Obrázek 4.1: Vektor shody

Pokud hledaný prefix v databázi existuje, pak určitě bude zachycen ve vektoru shody, protože filtry nikdy nemohou chybně označit dříve vložený prvek jako cizí. Dále také platí, že v tabulce nemusí být zastoupeny prefixy všech délek a jejich počet, W_{dist}, může být menší než W. V tom případě prázdné Bloomovy filtry nikdy nepřispějí svým výsledkem do vektoru shody. Rozptylovací tabulky pro jednotlivé délky prefixu jsou poté prozkoumány vzestupně od nejdelšího prefixu hlásícího shodu k nejkratšímu. Pokud je prvek ve vektoru shodu nastaven, ale rozptylovací tabulka příslušející dané délce prefixu adresu nenalezne, pak se jednalo o chybu. Algoritmus končí, pokud je nalezena shoda nebo pokud byly prohledány všechny délky. Výsledkem porovnání bude next hop adresa.

Stejného přístupu lze navíc využít při nasazení IPv4 i IPv6 protokolu.

4.1 Důsledek chyb prvního druhu

Důležitým parametrem ovlivňujícím výkon celého algoritmu je průměrný počet přístupů do rozptylovací tabulky, který označíme jako E_{avg}. Počet těchto přístupů potřebných k určení správné délky prefixu pro danou IP adresu je dán počtem filtrů hlásících shodu. Předpokládejme, že všechny filtry mají stejnou pravděpodobnost chyby f. Nechť dále B_l označuje počet filtru pro délky prefixu vetší než l. Potom pravděpodobnost, že právě i filtrů zpracovávající prefixy délky vetší než l, chybně označí adresu za svou, je dána vztahem:

$$P_l = \binom{B_l}{i} f^i (1-f)^{B_l - i} \tag{4.1}$$

Pro každou hodnotu i bychom potřebovali dalších i přístupů do rozptylovací tabulky. Očekávaný počet "zbytečných"přístupů pro vyhledání prefixu délky l je:

$$E_l = \sum_{i=1}^{B_l} i \binom{B_l}{i} f^i (1-f)^{B_l - i} \tag{4.2}$$

Což odpovídá binomickému rozložení, pro jehož střední hodnotu platí:

$$E_l = B_l f \tag{4.3}$$

Z výše uvedené rovnice vyplývá, že očekávaný počet nadbytečných přístupů do tabulek, pro prefixy jisté délky, je roven součinu počtu filtrů pro všechny delší prefixy a pravděpodobnosti dopuštení se chyby. Chápeme-li B jako celkový počet filtru v systému, pak pro nejhorší případ E_l, který označíme jako E_{add} platí:

$$E_{add} = Bf \tag{4.4}$$

Což je maximální počet nadbytečných přístupů do tabulek způsobený možností výskytu chyby při procesu vyhledání, bez ohledu na to, jaká adresa je na vstupu. A protože se jedná pouze o přístupy navíc, tak průměrný očekávaný počet přístupů pro adresy, jež v systému nejsou, bude:

$$E_{avgmin} = E_{add} = Bf \tag{4.5}$$

Pro adresy, jež v systému určite jsou, platí:

$$E_{avgmax} = (B - 1)\, f + 1 \tag{4.6}$$

Horní mez pro libovolnou adresu je zcela určitě:

$$E_{avg} = E_{add} + 1 = Bf + 1 \tag{4.7}$$

kde je navíc přičten přístup pro prefix hledané délky. Není-li prefix v systému, pak lze tento přístup navíc chápat jako shodu generovanou filtrem pro prefix nulové délky, jež se bude vždy shodovat. V nejhorším možném případě dojde k situaci, že vlivem chyb IP adresa způsobí shodu ve všech filtrech. Pro tento stav je počet maximální přístupů roven:

$$E_w = B + 1 \tag{4.8}$$

Nebudeme-li uvažovat prefix délky 0, tedy takový, který se bude vždy shodovat, pak lze tento prípad zredukovat na:

$$E_w = B \tag{4.9}$$

V nejhorším možném případě, ve zde popsané konfiguraci, B odpovídá W_{dist} přístupů pro vyhledání adresy. Zatímco rovnice 4.7 vyjadřuje očekávaný průměrný počet přístupů pro vyhledání nejdelšího shodného prefixu, tak rovnice 4.8 ukazuje nejhorší možný případ. Z čehož vyplývá, že další možná vylepšení by se mohla soustředit právě na redukci celkového počtu filtrů v systému a tedy i na omezení nejhoršího možného případu počtu přístupů do paměti. Několik návrhů pro urychlení algoritmu a snížení nejhoršího počtu přístupů je představeno v kaptiole7.

4.2 Proč je důležité omezit počet přístupů do paměti?

Pokud budeme uvažovat nasazení na nějakém směrovači, pak by filtry pravděpodobně byly uloženy v drahé a rychlé SRAM paměti, zatímco čítače, spojené s jednotlivými bity Bloomových filtrů, a rozptylovací tabulky by byly uloženy v pomalejší a daleko levnější hlavní paměti DRAM.

Průměrná paměť typu DRAM má přístupovou dobu 60-100 ns [8] [10]. Pro SRAM lze najít hodnoty okolo 10-20 ns[13] [1]. Z těchto hodnot tedy vyplývá, že přístupy do hlavní paměti do rozptylovací tabulky jsou několikanásobně pomalejší a je třeba je co nejvíce omezit.

Kapitola 5

Konfigurace a optimalizace

Seznam proměnných charakterizující systém, některé z nich byly popsány již dříve:

N, zamýšlený počet prefixů podporovaný programem

M, paměť potřebná pro uložení Bloomových filtrů

W_{dist}, podporovaný počet rozdílných délek prefixů

m_i, velikost jednotlivých filtrů

k_i, počet rozptylovacích funkcí jednotlivých filtrů

n_i, počet prefixů uložených v jednotlivých filtrech

I přesto, že práce je schopná pracovat s adresami IPv6, pro názornost budeme v následujícím textu, nebude-li uvedeno jinak, pracovat pouze s IPv4. Dle statistik, současné BGP směrovací tabulky přesahují 100 000 záznamů, avšak mohou obsahovat záznamů mnohem více (viz. [2]). Proto, nebude-li uvedeno jinak, budeme předpokládat $N = 250000$, abychom zajistili dostatečně velkou kapacitu směrování.

Dosud jsme předpokládali, že všechny filtry mají stejnou pravděpodobnost chyby f. Dokud je tato pravděpodobnost skutečně stejná, tak chování systému není ovlivněno rozložením délek jednotlivých prefixů. Nechť f_i označuje pravděpodobnost chyby i-tého filtru.

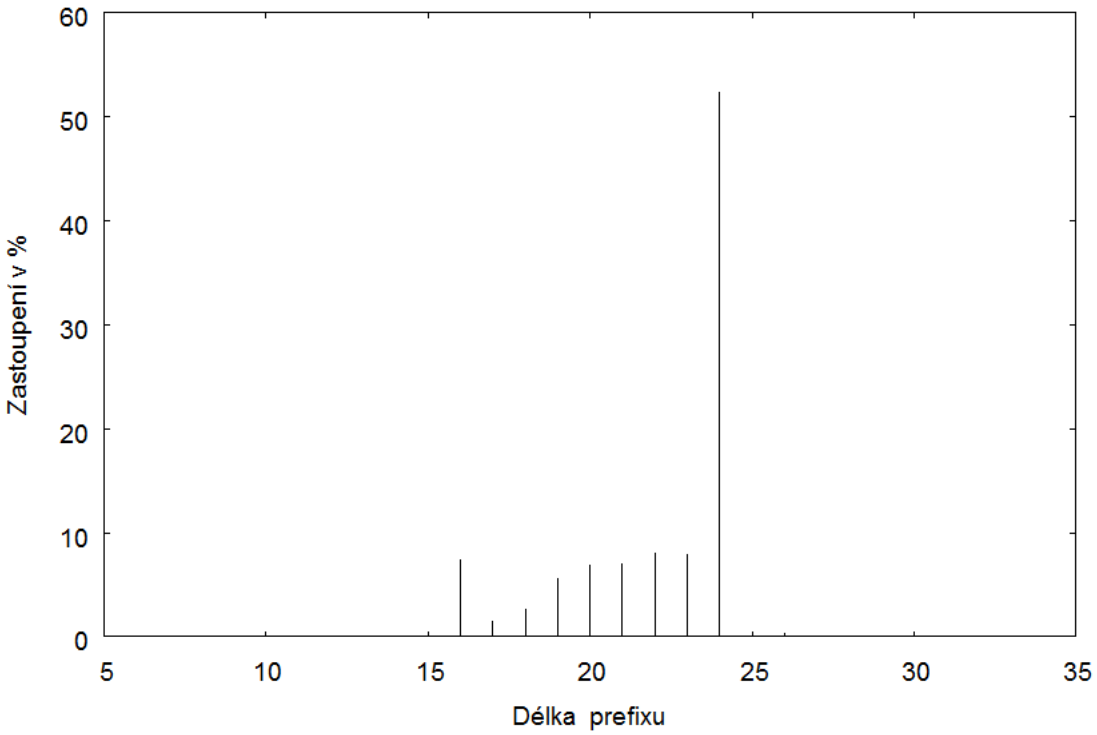
$$f_i = f = \left(\frac{1}{2}\right)^{\frac{m_i}{n_i}\ln 2} \quad \forall i \in \{1, \ldots, 32\} \tag{5.1}$$

Z čehož jednoznačně plyne:

$$\frac{m_1}{n_1} = \frac{m_2}{n_2} = \cdots = \frac{m_{32}}{n_{32}} = \frac{\sum m_i}{\sum n_i} = \frac{M}{N} \tag{5.2}$$

5.1 Asymetrické (nestejné) Bloomovy filtry

Z předchozích úvah vyplývá, že paměť by měla být alokována jednotlivým filtrům na základě jejich podílu na celkovém počtu prefixů. V případě symetrické konfigurace jednotlivých filtrů by každému filtru byla jednoduše vyčleněna stejná paměť $m = \frac{M}{B}$. Analýzou některých skutečných směrovacích tabulek však zjistíme, že rozložení délek prefixů má jisté charakteristiky. V případě protokolu IPv4 je převážná většina adres s prefixem délky 24 a pouze



Obrázek 5.1: Průměrné rozložení délek prefixů ve směrovacích tabulkách

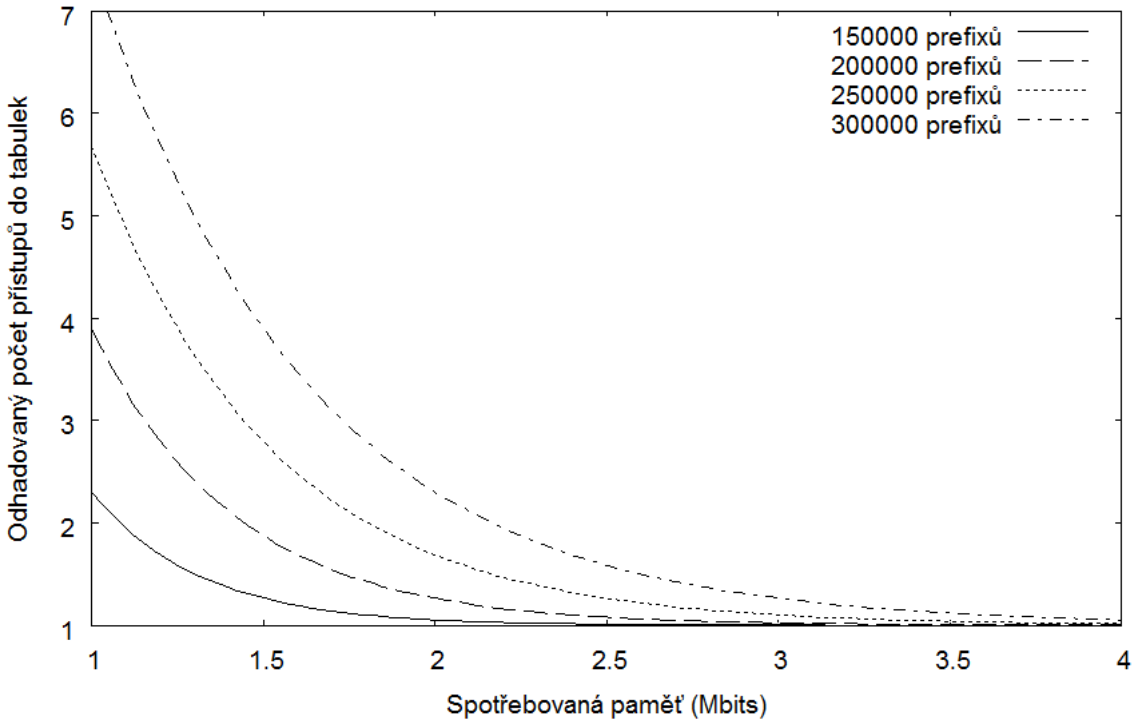
malý počet adres s prefixem méně než 8 bitů. Na obrázku 5.1 je ukázáno průměrné rozložení délek prefixů z několika snímků veřejně přístupné směrovací tabulky [2].

Pokud je systém nastaven rovnocenně pro všechny délky, avšak rozložení délek prefixů je jiné, pak dochází k tomu, že některé filtry mají alokovány velký nadbytek paměti, zatímco jiné nedostatek. To také znamená, že pravděpodobnost chyby jednotlivých filtrů je různá. Je tedy zřejmé, že pro zajištění stejné chybovosti všech filtrů je potřeba, aby paměť alokovaná jednotlivým filtrům byla rozdělena úměrně podle zastoupení dané délky prefixu v celkovém počtu. S tím by mohla souviset i případná změna počtu rozptylovacích funkcí, aby byla zachována minimální chyba. Tuto konfiguraci budeme označovat jako *asymetrické Bloomovy filtry*. Aplikací rovnice 4.7 na případ protokolu IPv4 vyjádříme očekávaný počet přístupů jako:

$$E_{avg} = 32 \times \left(\frac{1}{2}\right)^{\frac{M}{N}\ln 2} + 1 \tag{5.3}$$

V grafu zobrazeném na Obr. 5.2 je vykreslena závislost očekávaného průměrného počtu přístupů, E_{avg}, na celkové paměti, M, pro různé hodnoty N. Pro uložení 250000 prefixů a spotřebě zhruba 2Mb paměti je průměrný počet přístupů menší než dva. Můžeme tvrdit, že takovýto systém je paměťově úsporný, neboť vyžaduje pouze 8 bitů na jeden prefix. Zdvojnásobením použité paměti téměř dosáhneme optimálního jednoho přístupu.

Dosazením do rovnice 4.8 vyplyne, že nejhorší možný počet přístupů je roven 33, který ovšem může být zredukován podle 4.9 na hodnotu 32.

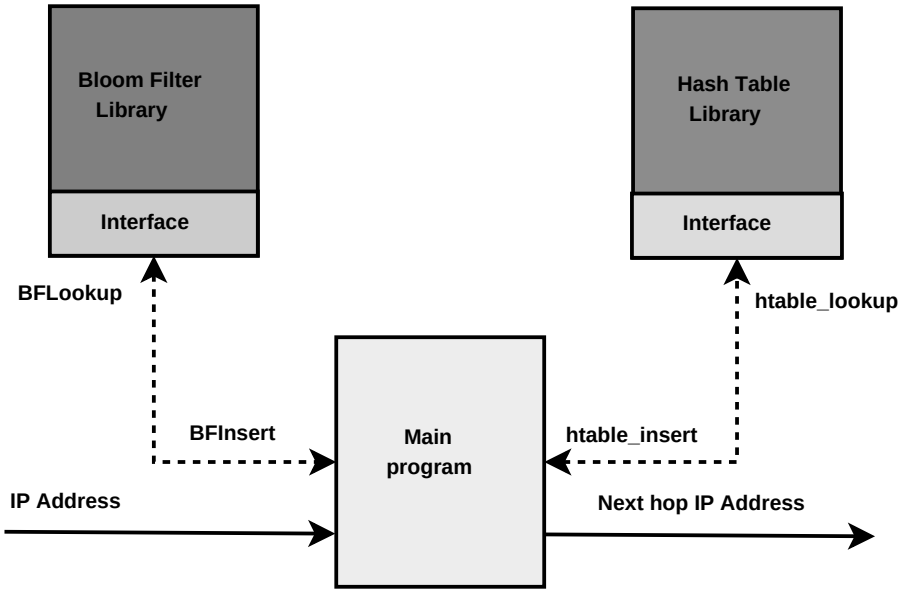


Obrázek 5.2: Očekávaný počet přístupů, do rozptylovací tabulky v závislosti na celkové paměti M zabrané filtry pro různá N. V konfiguraci pro IPv4 využívající 32 asymetrických Bloomových filtrů

Kapitola 6

Popis implementace

Celá práce je napsána v jazyce C a implementace je rozdělena do třech pomyslných, do jisté míry nezávislých, částí. Především je to hlavní program, který řídí načítání dat a jejich zpracování, dále knihovna implementující práci s rozptylovací tabulkou a vlastní implementace Bloomova filtru.



Obrázek 6.1: Návrh implementace a komunikace mezi jednotlivými částmi

6.1 Parametry programu

`-r, --routing-table <filename>`
Specifikuje soubor představující směrovací tabulku.

`-i, --input <filename>`
Specifikuje soubor obsahující adresy cílů (simulující cílové adresy příchozích paketů).

`-o, --output <filename>`
Volitelný parametr. Specifikuje výstup. Implicitně se jedná o stdout.

`-f, --false-positive <probability>`
Specifikuje požadovanou hodnotu chyby. Platné hodnoty leží v intervalu (0, 1).

`-e, --expected-count <elements>`
Očekávaný počet prvků, které budou Bloomovy filtry muset reprezentovat. Jedná se o celkový počet prvků, který bude muset systém celkově pojmout.

`-p, --ip-version <4|6>`
Přepíná mezi použitím IPv4 nebo IPv6 adres.

`-a, --asymmetric-filters`
Upraví očekávané počty prvků pro jednotlivé filtry dle analýz skutečných směrovacích tabulek.

`-h, --help`
Vytiskne popis jednotlivých parametrů.

`-v, --version`
Vytiskne jméno autora a verzi programu.

6.2 Formát vstupu a výstupu

Směrovací tabulka

Každý řádek souboru odpovídá jednomu směrovacímu záznamu. Očekávaný tvar je následující:

```
adresa-sítě/délka-prefixu next-hop
```

kde `adresa-sítě` je síťová adresa protokolu IPv4 nebo IPv6, jenž může být správně přečtena funkcí `inet_ntop()`, `délka-prefixu` je číslo označující délku prefixu síťové adresy a `next-hop` označuje adresu následujícího skoku, opět ve formátu rozpoznatelného funkcí `inet_ntop()`. Na počtu mezer oddělujících jednotlivé části nezáleží.

Adresy příchozích paketů

Formát souboru je tvořen řádky ve tvaru:

```
ip-adresa
```

kde `ip-adresa` je platná adresa protokolu IPv4 nebo IPv6 rozpoznatelná funkcí `inet_ntop()`. Tato adresa má význam cílové adresy jednoho příchozího paketu.

Výstup programu

Za každý příchozí paket je výstupem jeden řádek, jehož formát je následující:

```
ip-adresa -> next-hop
```

kde `ip-adresa` je cílová adresa paketu a `next-hop` je výsledná adresa, na kterou bude paket odeslán. V případě, že žádný záznam není nalezen je `next-hop` adresa nahrazena textem `Not found!`.

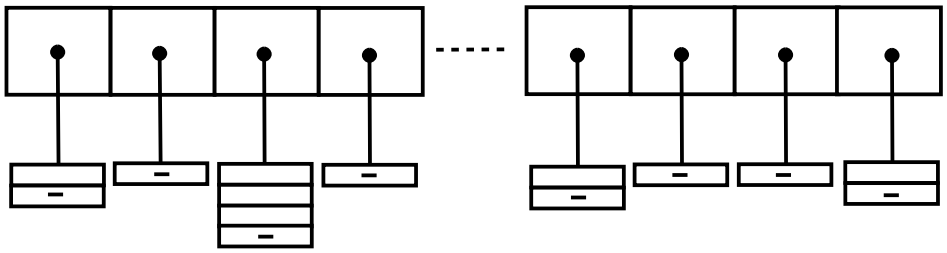
Konfigurační soubor

Program má v sobě zabudovánu distribuci rozdělení prefixů odpovídající grafu 5.1. Je-li však přítomen soubor `./config/asym.cfg` v relativní cestě od spustitelného souboru programu, pak bude rozdělení načteno z něj. Tento konfigurační soubor obsahuje pouze dva řádky, jeden popisuje rozdělení pro IPv4 a druhý rozdělení pro IPv6. Jejich syntax je následující:

```
IPV4=hodnota-pro-prefix1 hodnota-pro-prefix2 ...
IPV6=hodnota-pro-prefix1 hodnota-pro-prefix2 ...
```

6.3 Knihovna pro práci s rozptylovací tabulkou

Tabulka je v podstatě tvořena polem jednosměrně vázaných lineárních seznamů, které řeší případné kolize tj. mapování různých hodnot na stejnou pozici v tabulce. Položkami seznamů jsou síťová a next hop adresa.



Obrázek 6.2: Ukázka rozptylovací tabulky

Pohyb v tabulce je implementován po vzoru standardní knihovny C++ pomocí iterátorů. Pro práci s nimi je možno použít funkce `htable_begin`, `htable_end` a `htable_next`, které, jak jejich názvy napovídají, slouží k získání iterátoru na první, poslední, resp. následující položku. Dalšími operacemi zahrnující práci s iterátory je jejich porovnání a zpřístupnění záznamu, na který ukazují. Toto zajišťují funkce `htable_it_eq` a `htable_it_dereference`.

Vyhledávání a vkládání do tabulky je zprostředkováno dvěma podobnými funkcemi, jimiž jsou `htable_lookup`, resp. `htable_insert`. Obě tyto funkce se vkládaný prvek pokusí nejprve vyhledat, ale při operaci vkládání je v případě nenalazení nový prvek vložen, při pouhém vyhledání nikoliv.

Pro porovnávání adres při vyhledávání slouží funkce `addrcmp`, která testuje dvě adresy na shodu. Funkce přijímá parametr - délku prefixu, který zajistí, že se budou porovnávat pouze bity příslušných síťových částí adres.

6.4 Bloomův filtr

Informacemi nutnými k vytvoření Bloomova filtru je tolerovaná pravděpodobnost chyby a očekávaný počet prvků. Z těchto informací si každý filtr spočítá velikost bitového pole a nutný počet rozptylovacích funkcí. Protože směrovací informace nejsou statické, ale mění se v čase, jedná se o filtry počítané umožňující odstranění směrovacího záznamu. Implementovaná struktura Bloomova filtru zahrnuje následující položky:

- `prefix` - délka reprezentovaných prefixů daného filtru

- `k` - počet rozptylovacích funkcí
- `fpp` - zadaná false positive
- `m` - velikost bitoveho pole
- `n` - skutečný počet prvků v poli
- `a` - bitové pole
- `seed` - pole různých semínek pro rozptylovací funkci
- `counter` - pole čítačů

Rozptylovací funkce

Vzhledem k tomu, že každý filtr si sám odvozuje nutný počet rozptylovacích funkcí a knihovna pro práci s Bloomovými filtry obsahuje pouze jednu, je nezbytné zajistit požadovaný počet různých funkcí jinak. Toho je docíleno volbou vhodných unikátních počátečních *semínek* příslušejících jednotlivým rozptylovacím funkcím.

6.5 Vyhledání nejdelšího shodného prefixu

Operace vyhledání nejdelšího shodného prefixu je implementována v souladu s principem popsaným v kapitole 4. Po inicializaci všech rozptylovacích tabulek a Bloomových filtrů dochází ke zpracování směrovací tabulky - tj. načtení každého prefixu a jeho next hop adresy.

Poté následuje čtení cílových adres ze souboru. Tyto adresy představují vstupy dotazů na Bloomovy filtry. Odpovědi všech filtrů jsou poté seskupeny do pole - vektoru shody. Pro nejdelší shodující se prefix je přistoupeno do jeho rozptylovací tabulky, aby mohla být získána požadovaná adresa následujícího skoku. Jak bylo již zmíněno, pokud se prefix v tabulce nenachází, jednalo se o chybnou shodu a stejný postup se opakuje pro další (kratší) délku prefixu. Toto probíhá dokud není nalezena skutečná shoda nebo nezbývá žádný filtr hlásící shodu. Posledním krokem je tisk některých statistických údajů a uvolnění zdrojů.

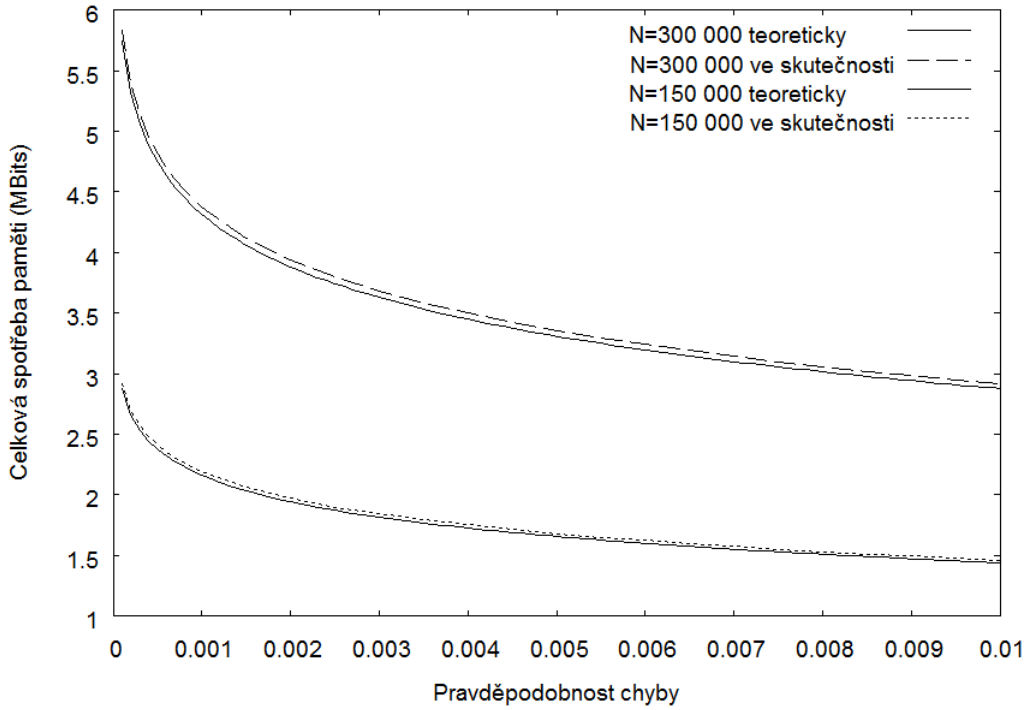
Kapitola 7

Výsledky

V této kapitole budou popsány výsledky experimentů s programem. Zjistíme skutečnou paměťovou složitost a počty dotazů na rozptylovací tabulky. Na konci kapitoly nastíníme možné úpravy našeho algoritmu, díky kterým docílíme omezení nejhoršího možného počtu přístupů do paměti.

7.1 Paměťová složitost

Bloomovy filtry jsou významné svojí nevelkou paměťovou náročností, zjistíme tedy, kolik paměti je skutečně potřeba pro danou směrovací tabulku s danou nastavenou tolerovanou pravděpodobností chyby. Z výše uvedených závislostí vyplývá, že skutečné hodnoty na-



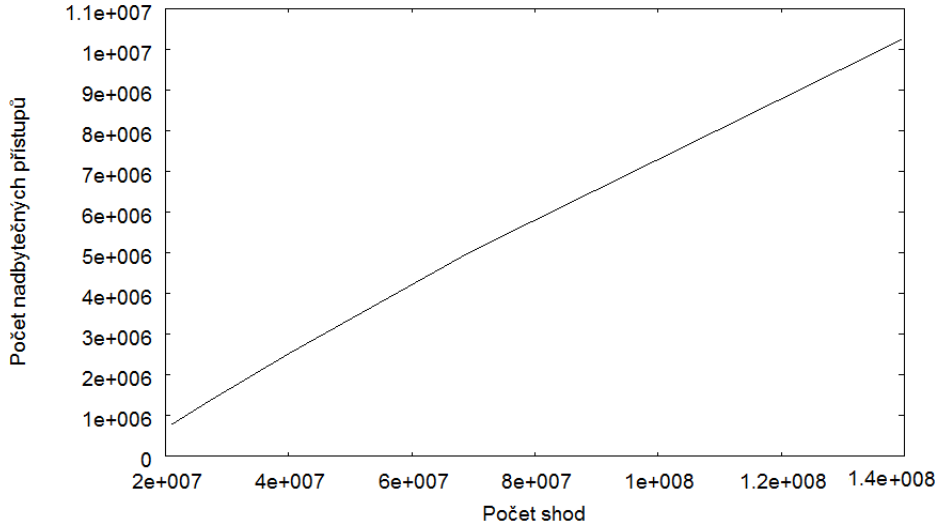
Obrázek 7.1: Porovnání teoretické a skutečné hodnoty spotřebované paměti pro různá N.

měřené hodnoty spotřebované paměti kopírují teoreticky vypočtené závislosti. Je patrné,

že skutečné hodnoty jsou malinko vyšší, to je způsobeno tím, že jsem do spotřebované paměti zahrnul celou strukturu Bloomova filtru, která je popsána v předchozí kapitole, a nikoliv pouze jeho bitový vektor.

7.2 Počet přístupů do rozptylovacích tabulek

Nejprve prozkoumáme závislost mezi počtem shod, které filtry vygenerovaly ve vektoru shody a počtem nadbytečných přístupů do tabulek. Pro tento a další test jsem vytvořil tři sady náhodně vygenerovaných IP adres. Každá z těchto sad obsahuje milión adres. Výsledné hodnoty jsou průměrem těchto z těchto třech sad.



Obrázek 7.2: Závislost mezi počtem nadbytečných přístupů a celkovým počtem shod hlášených jednotlivými filtry.

Dle očekávání jsme zjistili, že čím více shod je vygenerováno, tím větší je šance, že se dopustíme zbytečného dotazu na rozptylovací tabulku pro danou délku prefixu.

Testovací sada I - Náhodně generovaný vstup

Dále se zaměříme na skutečné průměrné počty přístupů do tabulek v závislosti na celkové zabrané paměti, kterou ovlivňuje očekávaný počet prefixů N a akceptovatelná pravděpodobnost chyby f. Teoretické hodnoty byly vypočteny užitím vztahu 5.3. V posledním sloupci tabulek je nejhorší zaznamenaný počet dotazů na rozptylovací tabulku při symetrické/asymetrické konfiguraci. Při tomto testu byly použity náhodně generované IP adresy, stejně jako v předchozím případě. Výsledné hodnoty z tohoto testu jsou zapsány v tabulkách 7.1, 7.2 a 7.3.

Testovací sada II - Platné adresy

V tomto případě byly použity stejné tabulky jako v předchozím testu, ale vstupem byly IP adresy generované ze směrovací tabulky, tedy s téměř očekávaným rozložením délek prefixů. I tato sada obsahovala milión adres. Výsledné hodnoty z tohoto testu jsou zapsány v tabulkách 7.4, 7.5 a 7.6.

Paměť(Mb)	N	f	Symetrické	Asymetrické	Teoreticky	Nejhorší
6.097680	300000	0.0001	2.537737	1.185130	1,001837	6/3
4.390080	300000	0.001	2.827133	1.337103	1.028296	6/5
3.951240	300000	0.002	2.972844	1.457824	1.057142	7/5
3.512288	300000	0.004	3.164144	1.591934	1.115414	7/6
3.073464	300000	0.008	3.372317	1.819792	1.233063	7/6
7.113688	350000	0.0001	2.262127	1.076390	1,001837	5/3
5.050880	350000	0.001	2.459428	1.169274	1.031188	6/4
4.545792	350000	0.002	2.578961	1.244079	1.062388	6/4
4.040960	350000	0.004	2.719052	1.338766	1.124758	6/5
3.535872	350000	0.008	2.909252	1.472856	1.249565	6/6

Tabulka 7.1: Hodnoty počtů přístupů pro směrovací tabulku s 318414 záznamy

Paměť(Mb)	N	f	Symetrické	Asymetrické	Teoreticky	Nejhorší
3.031040	150000	0.0001	2.811261	1.143739	1.001945	6/4
2.165504	150000	0.001	2.823482	1.348974	1.031104	6/5
1.948928	150000	0.002	2.955870	1.446954	1.062242	7/5
1.732608	150000	0.004	3.157514	1.620154	1.124451	7/6
1.516032	150000	0.008	3.388212	1.849671	1.249039	7/8
4.041472	200000	0.0001	2.092167	1.026330	1.001944	5/2
2.886912	200000	0.001	2.271391	1.093877	1.031136	5/4
2.598400	200000	0.002	2.356001	1.134985	1.062268	5/4
2.309888	200000	0.004	2.464847	1.198314	1.124528	6/4
2.021376	200000	0.008	2.619071	1.288012	1.249039	6/5

Tabulka 7.2: Hodnoty počtů přístupů pro směrovací tabulku s 159207 záznamy

Paměť(Mb)	N	f	Symetrické	Asymetrické	Teoreticky	Nejhorší
1.516032	75000	0.0001	2.436771	1.142520	1.001938	6/4
1.098528	75000	0.001	2.830640	1.375322	1.028114	6/5
0.988752	75000	0.002	2.978916	1.456379	1.056798	7/5
0.879040	75000	0.004	3.139559	1.616023	1.114701	7/6
0.769344	75000	0.008	3.396975	1.848952	1.231607	7/7
2.020608	100000	0.0001	2.090301	1.023056	1.001945	5/2
1.464304	100000	0.001	2.265974	1.101187	1.028168	5/3
1.317968	100000	0.002	2.348827	1.138566	1.056899	6/4
1.171656	100000	0.004	2.462844	1.212358	1.114920	6/4
1.025320	100000	0.008	2.607472	1.310224	1.232133	6/5

Tabulka 7.3: Hodnoty počtů přístupů pro směrovací tabulku s 79604 záznamy

Z uvedených hodnot vidíme, že při stejné velikosti spotřebované paměti asymetrická konfigurace poskytuje v naprosté většině případů lepší výsledky než konfigurace se stejným nastavením všech filtrů.

Dále je zřejmé, že méně paměti pro Bloomovy filtry má za následek zhoršený průměrný počet přístupů do rozptylovacích tabulek. Stejný závěr platí i pro jejich nejhorší pozorované případy, které zpravidla dosahovaly nejmenších hodnot při nastavení s nejvyšší uvedenou

Paměť(Mb)	N	f	Symetrické	Asymetrické	Teoreticky	Nejhorší
4.314880	300000	0.001	1.675211	1.166510	1.028296	6/4
3.881984	300000	0.002	1.708897	1.243140	1.057142	6/5
3.449088	300000	0.004	1.754717	1.322066	1.115414	6/5
3.016448	300000	0.008	1.801173	1.472505	1.233063	6/5
5.033472	350000	0.001	1.582450	1.087336	1.031188	6/3
4.528640	350000	0.002	1.613530	1.118972	1.062388	5/3
4.023552	350000	0.004	1.647349	1.187842	1.124758	6/4
3.518720	350000	0.008	1.691643	1.270877	1.249565	6/4

Tabulka 7.4: Hodnoty počtů přístupů pro směrovací tabulku s 318414 záznamy II

Paměť(Mb)	N	f	Symetrické	Asymetrické	Teoreticky	Nejhorší
3.049624	150000	0.0001	1.650087	1.076189	1.001945	5/3
2.287480	150000	0.001	1.726268	1.190054	1.031104	6/4
2.058056	150000	0.002	1.770639	1.246612	1.057142	6/4
1.828616	150000	0.004	1.814607	1.348814	1.115414	6/5
1.599272	150000	0.008	1.876842	1.488091	1.233063	7/5
4.065640	200000	0.0001	1.528474	1.011349	1.028296	4/2
3.049496	200000	0.001	1.570133	1.041927	1.031188	5/3
2.743584	200000	0.002	1.594592	1.067243	1.062388	5/3
2.437760	200000	0.004	1.627072	1.105529	1.124758	5/3
2.131872	200000	0.008	1.664314	1.159512	1.249565	6/4

Tabulka 7.5: Hodnoty počtů přístupů pro směrovací tabulku s 159207 záznamy II

Paměť(Mb)	N	f	Symetrické	Asymetrické	Teoreticky	Nejhorší
1.525496	75000	0.0001	1.653922	1.088474	1.001938	5/3
1.144392	75000	0.001	1.734384	1.188069	1.028114	6/4
1.029640	75000	0.002	1.772462	1.240662	1.056798	6/4
0.914952	75000	0.004	1.814056	1.325138	1.114701	6/3
0.800280	75000	0.008	1.873465	1.479475	1.231607	6/3
2.033616	100000	0.0001	1.527705	1.012521	1.001945	4/3
1.525456	100000	0.001	1.571552	1.045088	1.028168	4/3
1.372504	100000	0.002	1.594870	1.063668	1.056899	4/3
1.219544	100000	0.004	1.623539	1.104987	1.114920	5/3
1.066600	100000	0.008	1.668946	1.168158	1.232133	5/3

Tabulka 7.6: Hodnoty počtů přístupů pro směrovací tabulku s 79604 záznamy II

spotřebovanou pamětí a opačně.

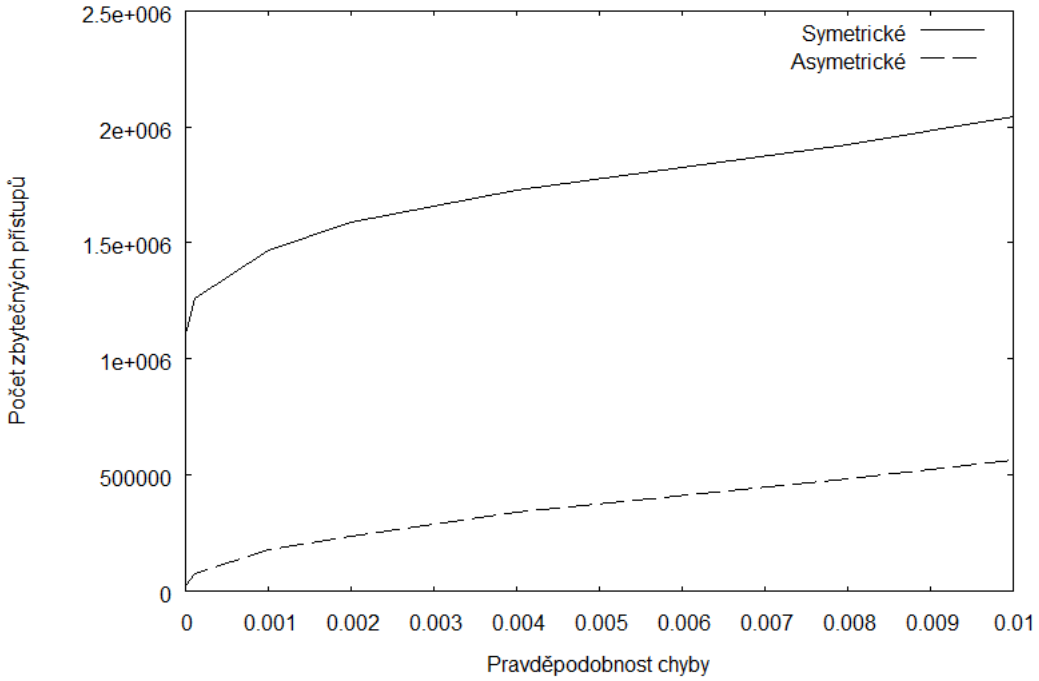
I přes skutečnost, že dle rovnice 4.9 je teoretická hodnota nejhorších možného počtu dotazů na rozptylovací tabulku během jednoho vyhledání rovna 32, nebyla během testu zpravidla zjištěna hodnota vyšší než 7 přístupů.

Při testu s platnými adresami byly naměřené počty přístupů prokazatelně nižší než u testu s náhodně generovanými adresami, což je v rozporu s výsledky prezentovanými v jiných publikacích[6], kde tvrdí, že při testech s náhodně generovanými vstupními IP adresami nedošlo k žádnému výskytu chyby prvního druhu a naopak maximální chybovost

se projevila u testu s adresami vytvořenými z reprezentované směrovací tabulky.

7.3 Srovnání počtu zbytečných přístupů

Porovnáme, kolik je zbytečných přístupů, když ponecháme stejnou paměť (nastavení) pro obě metody. Na Obrázku 7.3 je zachycen naměřený počet chybně určených délek prefixů pro mou nejrozsáhlejší tabulku pro různé hodnoty pravděpodobnosti chyby. Obě konfigurace byly spuštěny s parametrem $N = 350000$.



Obrázek 7.3: Rozdíl mezi počtem nadbytečných přístupů obou konfigurací

Opět jasně vidíme, že asymetrická konfigurace poskytuje několikanásobně lepší výsledek. Z experimentů také vyplynulo, že při stejném nastavení je asymetrická konfigurace rychlejší než symetrická. Pro zajímavost, jsou v tabulce 7.7 uvedeny hodnoty získané pomocí programu `time` pro směrování jednoho milionu adres. Měření jsem provedl několikrát za různých podmínek, aby bylo vyloučeno vylepšení času druhého testu kvůli rychlým vyrovnávacím pamětem disku.

time	Symetrické	Asymetrické
real	0m20.623s	0m12.391s
user	0m14.005s	0m7.292s
sys	0m1.992s	0m1.976s

Tabulka 7.7: Porovnání délek běhů programu

Kapitola 8

Další možná vylepšení

Implementací asymetrických Bloomových filtrů může být při stejné spotřebě paměti zaručen téměř optimální průměrný výkon pro systémy s velkým počtem prefixů[6]. Avšak ani toto vylepšení neomezuje nejhorší možný počet dotazů na rozptylovací tabulky. Hlavním cílem zde uvedených úprav je snížení celkového počtu filtrů v systému právě za účelem omezení nejhoršího možného počtu přístupů do rozptylovacích tabulek na přijatelnou hodnotu.

8.1 Redukce filtrů reprezentující krátké prefixy

Z grafu průměrného rozložení délek prefixů na Obrázku 5.1 je patrné, že počty krátkých délek prefixů jsou téměř nebo zcela nulové. Bylo by tedy možné pro prvních a délek prefixů udržovat strukturu umožňující jejich přímé vyhledání. Tento způsob by poskytoval účinný způsob reprezentace krátkých prefixů a zároveň i snížení celkového počtu filtrů. Každá takto reprezentovaná délka prefixu znamená snížení nejhoršího možného počtu dotazů na rozptylovací tabulky o jedničku. Nyní se pokusme naznačit, jak by tohoto mohlo být dosaženo.

Trie (Prefixový strom)

Jedná se o stromovou datovou strukturu, která se používá pro uchovávání asociativního pole, kde klíčem bude v našem případě IP adresa. Na rozdíl od binárního vyhledávacího stromu, kde se podle hodnoty uzlu rozhoduje, do které větve dále sestoupit, trie v každém uzlu obsahuje všechny podsítě, kterými může pokračovat IP adresa v dosud prohledané cestě. Všichni následníci uzlu mají společný prefix, který je shodný s adresou přiřazenou k danému uzlu. Kořen je asociován s prázdným řetězcem. Hodnoty obvykle nejsou asociovány se všemi uzly, ale jen s listy a některými vnitřními uzly, které odpovídají klíčům.

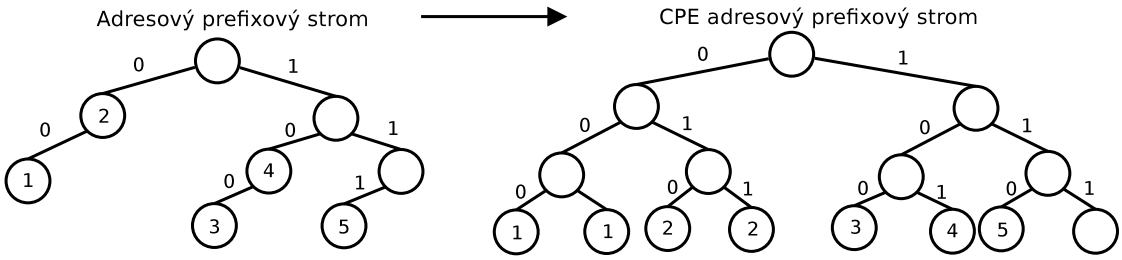
Uvažujme *binární trie* pro uložení prefixů IP adres, pro které platí, že jejich délka je menší nebo rovna a. S uzly stromu se pojí next hop adresy. Hrany jsou označeny hodnotou bitu na příslušném místě prefixu a cesta od kořene k uzlu tvoří jeho adresu. Poté provedeme řízené rozvinutí prefixů našeho stromu na délku rovnu a.

Controlled Prefix Expansion (Řízené rozvinutí prefixů)

Převádí množinu jistou množinu prefixů na ekvivalentní množinu prefixů s méně rozdílnými délkami prefixů[14], avšak za cenu jejich vyššího celkového počtu. Každý prefix v databázi je doplněn všemi možnými kombinacemi nul a jedniček, aby dosahoval požadované větší

délky. Je-li takto rozšířená adresa již v naší databázi, pak ji samozřejmě nepřidáme a ponecháme původní prefix. Bezmyšlenkovitá expanze však může přinést podstatné zvýšení paměťových nároků. Například problém vyhledání prefixu adresy lze převést na obyčejné prohledávání pole, rozšíříme-li všechny adresy na jejich plnou délku. Toto řešení by ovšem bylo nepřijatelné, protože v případě IPv6 protokolu by bylo zapotřebí 2^{128} prvků pole.

Na druhé straně může tato technika přispět k lepšímu celkovému výkonu především proto, že naším cílem je snížení počtu rozdílných délek W_{dist}.

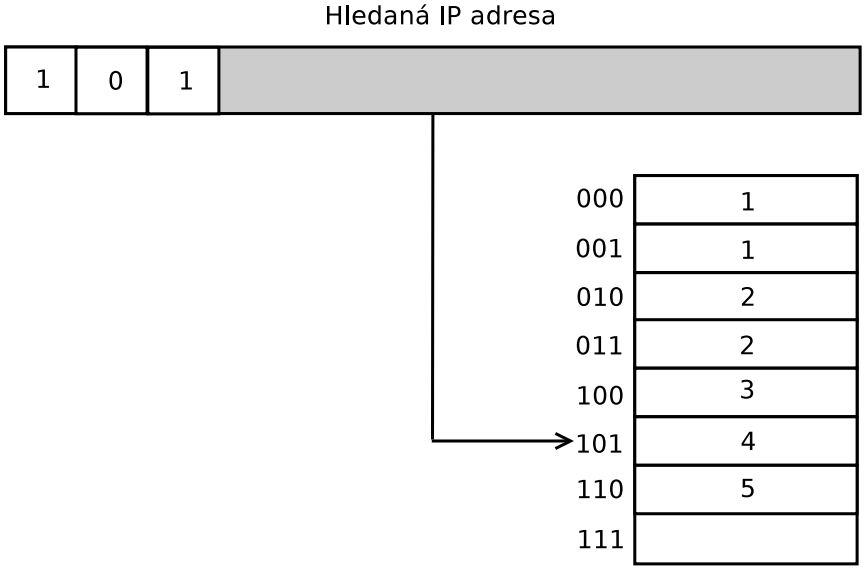


Obrázek 8.1: Použití techniky CPE

Vyhledávací tabulka pro přímé vyhledání krátkých prefixů

Nyní objasníme pojem a tvorbu této vyhledávací tabulky. Nejprve uložíme všechny prefixy, pro jejichž délku, l, platí $l <= a$, do binárního prefixového stromu. Poté provedeme Controlled Prefix Expansion (CPE) na výslednou délku prefix a, jak je naznačeno na Obrázku 8.1.

Next hop adresa, která je přiřazena každému listu v hloubce a, je uložena do tabulky na pozici, jež je dána bity na cestě v CPE prefixovém stromě od kořene k listu. Do tabulky je poté přistupováno pomocí prvních a bitů cílové IP adresy, které slouží jako index položky tabulky. Například výsledkem pro adresu s počátečními bity 101 by byl next hop 4, jak je ukázáno na Obrázku 8.2.



Obrázek 8.2: Použití vyhledávací tabulky pro směrování krátkých prefixů ($a = 3$)

Paměťové nároky této struktury jsou $2^a \times D_{nh}$ bitů, kde D_{nh} označuje počet bitů nutných k reprezentaci next hop adresy.

Existují různé metody nalezení optimálních parametrů CPE techniky [14]. Na základě vlastností získaných analýzou směrovacích tabulek bychom mohli zvolit $a = 20$. Z praktických (paměťových) důvodů budeme předpokládat, že next hop adresa představuje číslo výstupního portu směrovače. Pro směrovač s 256ti porty postačí k jednoznačné adresaci portu 8bitů. Tabulka s těmito parametry by spotřebovala 1MB paměti a Bloomovy filtry by tedy sloužili k reprezentaci prefixů délek 21...32. Výraz pro očekávaný průměrný počet přístupů do rozptylovacích tabulek se změní následovně:

$$E_{avg} = 12 \times \left(\frac{1}{2}\right)^{\frac{M \ln 2}{N - N_{20}}} + 1 \tag{8.1}$$

,kde N_{20} je počet prefixů s délkami menšími nebo rovny 20. Snížení počtu filtrů reprezentujících jednotlivé délky prefixů má také za následek snížení paměťových nároků pro dosažení optimálního průměrného výkonu.

8.2 Další snížení počtu filtrů

Snížení zbývajícího počtu Bloomových filtrů můžeme docílit dalším použitím řízeného rozvinutí prefixů. Volba vhodné délky rozvinutí záleží na konkrétním rozložení prefixů. V rozložení, které je ukázáno na Obrázku 5.1, převažují prefixy délky 21 až 24. Ty v průměru tvoří asi 70% celkového množství. Zaměříme-li se na delší prefixy, zjistíme, že jejich výskyt je poměrně ojedinělý. Podíl délek prefixů 25 až 32 tvoří méně než 2%.

Na základě těchto znalostí bychom mohli rozdělit ještě nepokryté prefixy do dvou skupin G_1 a G_2, jež obsahují množiny prefixů o délkách 21 - 24, respektive 25 - 32. Obě skupiny jsou rozvinuty na prefixy délky rovny maxima v dané skupině, tedy G_1 obsahuje pouze prefixy délky 24 a G_2 pouze délky 32. Nechť N_{24} značí velikost množiny v G_1 po rozvinutí a nechť N_{32} velikost množiny v G_2 po rozvinutí. Použití této techniky zvýší celkový počet prefixů v obou skupinách o *expanzní faktor* α_{24}, respektive α_{32}.

Výsledkem použití i této techniky by byly dva Bloomovy filtry a jedno vyhledávací pole s nejhorším možným výsledkem dvou přístupů do rozptylovacích tabulek a jednoho přístupu do pole. Vztah pro očekávaný průměrný počet přístupů do rozptylovacích tabulek tedy můžeme zapsat jako:

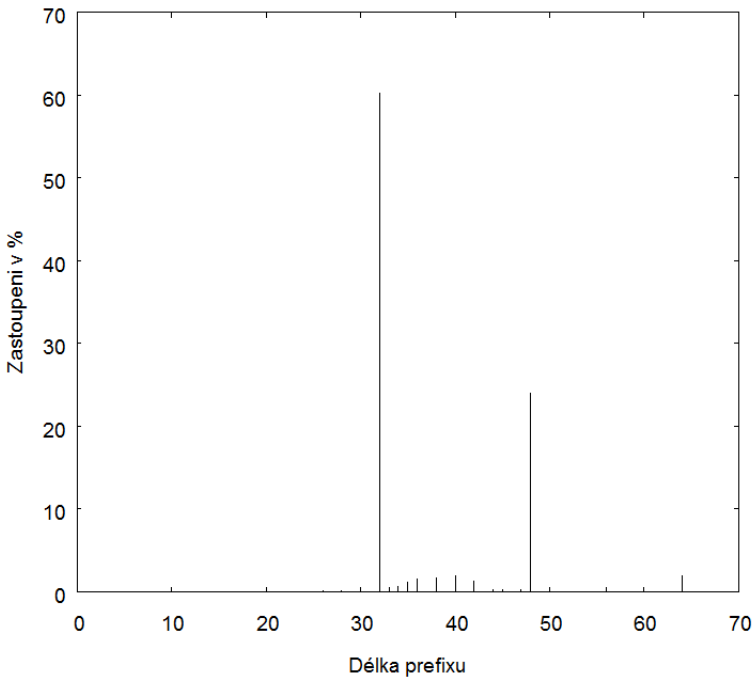
$$E_{avg} = 2 \times \left(\frac{1}{2}\right)^{\frac{M \ln 2}{\alpha_{24} N_{24} + \alpha_{32} N_{32}}} + 1 \tag{8.2}$$

Kapitola 9

IPv6

Téměř všechny zde uvedené principy lze uplatnit i v případě použití IPv6. Je vhodné mít na paměti, že i přes skutečnost čtyřikrát delších adres se Bloomovy filtry nijak nezmění, protože nejsou závislé na velikosti reprezentovaných prvků. Základní podmínkou použití našeho algoritmu pro IPv6 je udržitelný počet jedinečných délek prefixů.

Z pochopitelných důvodů je však potřeba změnit rozložení délek prefixů pro asymetrické nastavení. Průměrné rozložení vyplývající z analýzy dvou volně dostupných tabulek z [2] je na Obrázku 9.1. Počet prefixů v obou tabulkách se pohyboval okolo 2800. Významným zjištěním je, že počet jedinečných délek prefixů je 34, což v podstatě odpovídá nejhoršímu případu u IPv4.



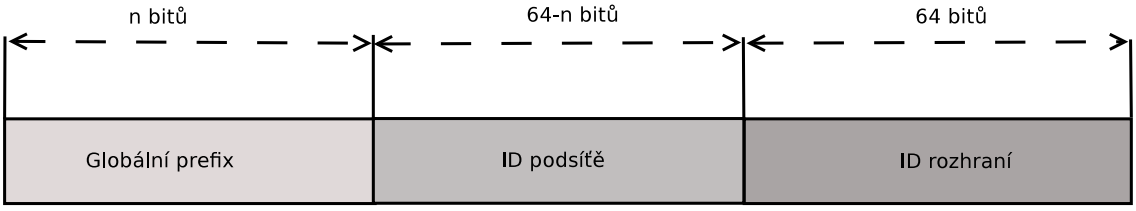
Obrázek 9.1: Průměrné rozložení délek IPv6 prefixů

Dále se budeme věnovat úvahám, zda je možné očekávat razantní zvýšení počtu jedinečných délek prefixů či nikoliv.

9.1 Architektura IPv6 adres

Formát a rozdělení IPv6 adres je uvedeno v RFC 4291 [12]. Nejpočetnější skupinou adres, která nejvíce ovlivňuje počet rozdílných délek prefixů, jsou individuální globální adresy (*global unicast*). Tyto adresy mohou být agregovány, podobně jako IPv4 adresy využívající CIDR.

Individuální globální adresa se skládá ze tří částí, které jsou zobrazeny na Obrázku 9.2. Jedná se o *globální prefix*, *identifikaci podsítě* a *identifikátor rozhraní*. Kromě speciálních



Obrázek 9.2: Struktura IPv6 adresy

adres, které začínají 000, musí všechny tyto adresy mít 64bitový identifikátor rozhraní ve formátu Modified EUI-64. Z čehož vyplývá, že identifikace podsítě zabírá spolu s globálním prefixem dohromady také 64 bitů.

Individuální globální adresy začínající na 000 nemají žádné požadavky na délku identifikátoru rozhraní. A jelikož se jedná o speciální adresy určené k tunelování IPv6 přes IPv4, tak nepředpokládáme, že by nějak výrazně přispěly k vyššímu počtu jedinečných délek prefixů.

9.2 Přidělování adres

Dle politik vydaných organizací Internet Assigned Numbers Authority (IANA) řídících počáteční rozdělování IPv6 adresového prostoru [7] je snahou přidělovat adresy hierarchicky, aby se byla umožněna snadná agregace směrovacích informací a bylo zamezeno zbytečnému růstu směrovacích tabulek. Dá se tedy předpokládat, že při dodržení těchto politik nebudou nedojde v budoucnu k výraznějšímu nárůstu odlišných délek prefixů.

Vzhledem k možným výrazným rozdílům mezi délkami prefixů by pravděpodobně nebylo vhodné aplikovat techniku řízeného rozvinutí prefixů (CPE) za účelem snížení celkového počtu filtrů.

Kapitola 10

Závěr

Práce ukázala způsob použití Bloomových filtrů při operaci hledání nejdelšího shodného prefixu, která hraje klíčovou roli při směrování. Hlavním úkolem Bloomových filtrů je zúžení možného rozsahu vyhledávání, tj. správná identifikace délky prefixu a tedy i příslušné rozptylovací tabulky s adresou následujícího skoku.

Z důvodu lepšího využití paměti byla představena asymetrická konfigurace, která pro jednotlivé filtry rozděluje paměť podle očekávaného zastoupení jejich délky prefixu v celkovém počtu záznamů.

I přes skutečnost, že naměřené hodnoty se ukázaly býti vyšší než předpokládala teorie, myslím, že Bloomovy filtry nabízejí zajímavou možnost oproti klasickým přístupům. Vždy se jedná o kompromis mezi spotřebovanou pamětí a pravděpodobností chyby.

V uvedených rozšířeních byly nastíneny úvahy na další vylepšení, které by vedly k celkovému zrychlení směrování a omezení nejhoršího možného případu na dva dotazy na rozptylovací tabulky a jeden přístup do pole.

Úvahy o budoucím rozvoji IPv6 nás dovedli k závěru, že nasazení tohoto protokolu pravděpodobně nepovede k drastickému snížení výkonu, protože přidělování nových adres se děje hierarchicky s ohledem na možné velikosti směrovacích tabulek, a také proto, že Bloomovy filtry prakticky nezávisejí na velikosti "vkládaných" prvků. Implementovaný algorimus je tedy samozřejmě schopen pracovat s adresami protokolu IPv6, na který by měly být připraveny všechny nově vyvíjené síťové aplikace.

Literatura

[1] AllianceSemiconductor: CMOS SRAM datasheet. [Online].
URL `http://www.datasheetcatalog.org/datasheets2/12/128503_1.pdf`

[2] BGP Table Data. 2010, [Online].
URL `http://bgp.potaroo.net/.`

[3] Bloom, B. H.: Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 1970: s. 422–426, ISSN 0001-0782.

[4] Broder, A.; Mitzenmacher, M.: Network Applications of Bloom Filters: A Survey. In *Internet Mathematics*, 2002, s. 636–646.

[5] Chang Fay, G. S. e. a., Dean Jeffrey: Bigtable: A Distributed Storage System for Structured Data. 2006.

[6] Dharmapurikar, S.; Krishnamurthy, P.; Taylor, D. E.: Longest prefix matching using bloom filters. *IEEE/ACM Trans. Netw.*, 2006: s. 397–409, ISSN 1063-6692.

[7] IANA: IPv6 Address Allocation and Assignment Policy. 2002, [Online].
URL `http://www.iana.org/reports/2002/ipv6-allocation-policy-26jun02`

[8] Intel: 16-MBit DRAM-Interface flash memory datasheet. [Online].
URL `http://download.intel.com/design/archives/flash/docs/29053301.pdf`

[9] Matoušek, P.: *Studijní opora předmětu ISA*. FIT VUT v Brně, 2010.

[10] Micron: DRAM MT4LC4M16R6 datasheet. [Online].
URL `http://www.ece.umd.edu/courses/enee759h.S2003/references/D29_C.pdf`

[11] Pagh, A.; Pagh, R.; Rao, S. S.: An optimal Bloom filter replacement. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2005, ISBN 0-89871-585-7, s. 823–829.

[12] R. Hinden, S. D.: IP Version 6 Addressing Architecture. 2006, [Online].
URL `http://www.ietf.org/rfc/rfc4291.txt`

[13] Samsung: Synchronous Burst SRAM datasheet. [Online].
URL `http://www.datasheetcatalog.org/datasheets2/27/270351_1.pdf`

[14] Srinivasan, V.; Varghese, G.: Faster IP Lookups using Controlled Prefix Expansion. 1998.

Příloha A

Obsah DVD

Popis obsahu adresářů DVD disku přiloženého k této práci:

`/config` - ukázku konfigračního souboru

`/input` - sady cílových adres

`/rtables` - příklad směrovací tabulky

`/src` - zdrojový kód implementovaného programu

`/tex` - zdrojový text této práce v LaTeXu

`/thesis` - text této práce v pdf dokumentu